Pandas string functions

Create New Seiers with substrings

Create Raw string & Series with substrings

```
import pandas as pd
import numpy as np

# create Series of different string objects
s = pd.Series(['0', 'John Wood', 'Colin Welsh', 'my list', '02456', np.nan, 'HELLO WORLD', 'water%'])
```

```
# -----------------------------------------------------
# make a string
pat = r'[a-z]+' # raw string, used to search pattersns with reguklar expressions
pat
```

Inspect strings

Inspect dtype & len

```
# -----------------------------------------------------
# inspect str.
s.dtype          # here: dtype('O') - i.e mixed dtypes for each substring!
s.str.len()      # len of each string
   #.       0     1.0
   #.       .
   #.       5     NaN       # NaN has no len()
   #.       . ...
```

Inspect char type and content RETURN TRUE/FALSE

isalnum(), …

```
>>> s1 = pd.Series(['one', 'one1', '1', ''])
```

```
>>> s1.str.isalpha()
0     True
1    False
2    False
3    False
dtype: bool
```

```
>>> s1.str.isnumeric()
0    False
1    False
2     True
3    False
dtype: bool
```

```
>>> s1.str.isalnum()
0     True
1     True
2     True
3    False
dtype: bool
```

`Series.str.isalpha`
Check whether all characters are alphabetic.

`Series.str.isnumeric`
Check whether all characters are numeric.

`Series.str.isalnum`
Check whether all characters are alphanumeric.

`Series.str.isdigit`
Check whether all characters are digits.

`Series.str.isdecimal`
Check whether all characters are decimal.

`Series.str.isspace`
Check whether all characters are whitespace.

`Series.str.islower`
Check whether all characters are lowercase.

`Series.str.isupper`
Check whether all characters are uppercase.

`Series.str.istitle`
Check whether all characters are titlecase.

https://pandas.pydata.org/pandas-docs/stable/user_guide/text.html

Size/length Transformations

Divide substrings

Return 2 items
Only divided string parts

Return 3 items
string parts + separator

split(), r split()

```
"""
    SPLIT in more de
"""
# split no patterns
s = pd.Series(['Voyager starship', 'Enterprize', 'ger', np.nan])
s.str.split(expand=True) # by defaults splits spces!
```

```
           0         1
0    Voyager  starship
1 Enterprize      None
2        ger      None
3        NaN       NaN
```

separator – is removed with splt().

```
# with argument
s.str.split(pat="e", expand=True)
```

```
       0          1     2
0  Voyag  r starship  None
1    Ent      rpriz
2      g          r  None
3    NaN        NaN   NaN
```

Expand =True/false

```
# "from end", wiht "n" that limits the number of splits
s.str.rsplit(pat="e",n=1, expand=True)
```

```
            0           1
0       Voyag  r starship
1   Enterpriz
2           g           r
3         NaN         NaN
```

n = max nr of splits

```
# split onn special characters
s = pd.Series(["1+1=2"])
s.str.split(pat=r'\+|=', expand=True)
```

```
   0  1  2
0  1  1  2
```

partition(), rpartition()

- **expand** = True/False
- **pat =** REGEX

```
[212]: # Splits strin on the first occurence of pat
       s = pd.Series(["BlannnNnnnBla", "rNrese", np.nan])
       s.str.partition("n", expand=True) # Case sensitive
```

```
[212]:      0    1          2
       0  Bla    n  nnNnnnBla
       1  rNrese
       2  NaN  NaN        NaN
```

```
[211]: # ... last occurence of pat
       s.str.rpartition("n", expand=True)
```

```
[211]:            0    1       2
       0  BlannnNnn    n     Bla
       1                  rNrese
       2        NaN  NaN     NaN
```

Slice substrings

[index], slice(), get()

	if less char.	If NaNstri
• **s.str[m]**	NaN	NaN
• **s.str[m:n]**	returns max nr	NaN
• **s.str.slice(from,to)**	returns max nr	NaN
• **s.str.get(n)**	NaN	NaN

```
# ------------------------------------------------------------
# SLICE SUBSTRINGS

s = pd.Series(['A','Aaba', np.nan]);

# use index
s.str[2]
    #.         0    NaN    # if it passes last index >> NaN
    #.         1      b
    #.         2    NaN    3 just NaN

# slice with index
#        advantage: no NaN is substring is shorter then requested.
s.str[0:2]          # returns 2 first characters form all substrings in that series
s.str[0:20]         # -||- 20 ..., or the max nr of characters!

# str.slice()
s.str.slice(0,2)    # == s.str[0:2]
                    # returns a slice, 2 first characters of each substring, or les
                    # if there was less char, or NaN

# s.str.get()
s.str.get(1)       # 2nd char in each substr.
s.str.get(-1)      # last char in each substr.
s.str.get(10)      # return NaN is mising value
```

Join/Concatenate *strings/substrings*

cat() -Concatenate

cat()
- **na_rep** = False
- **sep**= "_ , ; …"
- **join** "inner" / "outer" / "left" …

Join substrings in one series

```
# eg
s1 = pd.Series(['a', 'b', np.nan ,'c', 'd'])

# turn substrings in one series into one long string
s1.str.cat( )                       # 'abcd'      # NaN are ignored!
s1.str.cat(sep=";_")                # 'a;_b;_c;_d'
s1.str.cat(sep="", na_rep="_")      # 'ab_cd'
```

Join Substrings in two series or in series & in lists

```
# combine substrings from two series or from series and list-like object
#.                  to make each substring longer
s1 = pd.Series(['a', 'b', 'c', 'd'])
s1.str.cat(["1", "2", "3", "4"])
    #.          0    a1
    #.          1    b2
    #.          2    c3
    #.          3    d4

#   CAUTION; IF YOU CONCATENATE SERIES WIHT INDEXED OBJ, EG LIST
#.          THE ORDER MAY CHNAGE AND JPOIN SUBSTRINGS WITH CRRESPONDING INDEXES
```

Join substrings in series/lists with diff. index and length

```
# eg:
s1 = pd.Series(['A', 'B', 'C'], index=[0,1,2])
s2 = pd.Series([str(x) for x in list(range(1,5))], index=[1,2,3,4])

# join="inner/ outer/ left/ right"
s1.str.cat(s2, join="outer", na_rep=" ---- ", sep=" ; ") # all items are added
s1.str.cat(s2, join="inner", na_rep=" ---- ", sep=" ; ") # only common are joined
s1.str.cat(s2, join="left", na_rep=" ---- ", sep=" ; ") # left/right
    #.          0      A ;  ----
    #.          1         B ; 1
    #.          2         C ; 2
```

join = "inner" / "outer" / "left" …
This decides which string will generate NaN in the other string

Content Modifications

Change Lower/upper cases

Lower(), upper(), capitalize(), swapcase(), casefold()

Series.str.lower
Converts all characters to lowercase.
Series.str.upper
Converts all characters to uppercase.
Series.str.title
Converts first character of each word to uppercase and remaining to lowercase.
Series.str.capitalize
Converts first character to uppercase and remaining to lowercase.
Series.str.swapcase
Converts uppercase to lowercase and lowercase to uppercase.
Series.str.casefold
Removes all case distinctions in the string.

Replace

Replace() & slice_replace()

- replace() : *replace pattern*
- slice_replace() first remove slice, then put something else instead

replace(from, to, "with")

```
# REPLCACE & slice_replcace

# replcace
s.str.replace( "%", "percent of beer in milk")  # with new value
s.str.replace( "water", "")                      # == remove

# Use regular expr.
s3 = pd.Series(['A', 'B', 'C', 'Aaba', 'Baca','dog'])
s3.str.replace('A\|dog', ' --blablaBleee-- ', case=True) # case sensitive, replcace "A" or "dog"
    #.        0        --blablaBleee--
    #.        1                      B
    #.        2                      C
    #.        3     --blablaBleee-- aba
    #.        4                   Baca
    #.        5        --blablaBleee--

# Caution
    # special char like $ must be with escape char before
    # eg: dollars.str.replace(r'-\$', '-')

# str.slice_replace()
s.str.slice_replace(0,2, "__________") # replace 0-2 char. in all substr's wiht a given val.
    #.        0           __________
    #.        1       __________hn Wood
    #.        2     __________lin Welsh
    #.        3         __________ list
    #.        4           __________456
```

YOU MAY USE FUNCTION TO REPLACE ITEMS But you must work on x.groups()

```
## REPLCACE with Function
s = pd.Series(["Buldog", "Maddog", "Catdog"])
s.str.replace('(\w{0,10}dog)', lambda x: x.groups()[0][0:3])
                                              # you must work on groups

0    Bul
1    Mad
2    Cat
dtype: object
```

Change copy number

repeat()

Single number repeats
repeat(repeats = n)

Each item (i) is repeated different # of x
repeat(repeats = [n1, n2, ... ni])

```
>>> s = pd.Series(['a', 'b', 'c'])
>>> s
0    a
1    b
2    c
dtype: object
```

Single int repeats string in Series

```
>>> s.str.repeat(repeats=2)
0    aa
1    bb
2    cc
dtype: object
```

Sequence of int repeats corresponding string in Series

```
>>> s.str.repeat(repeats=[1, 2, 3])
0      a
1     bb
2    ccc
dtype: object
```

Create paragraphs with defined length

wrap()

- expand_tabs = False
- replace_whitespace = True
- drop_whitespace = True
- break_long_words = False
- break_on_hyphens = False

width : *int,* max nr of characters in the line,

Other parameters define how to expand tabs(1st one) or deal with long worlds

Examples

```
>>> s = pd.Series(['line to be wrapped', 'another line to be wrapped'])
>>> s.str.wrap(12)
0             line to be\nwrapped
1    another line\nto be\nwrapped
dtype: object
```

Change ends

Add new ends

pad()

```
>>> s = pd.Series(["caribou", "tiger"])
>>> s
0    caribou
1      tiger
dtype: object
```

```
>>> s.str.pad(width=10)
0       caribou
1         tiger
dtype: object
```

```
>>> s.str.pad(width=10, side='right', fillchar='-')
0    caribou---
1    tiger-----
dtype: object
```

```
>>> s.str.pad(width=10, side='both', fillchar='-')
0    -caribou--
1    --tiger---
dtype: object
```

Pad()
- widht = , min width of resulting string; additional characters will be filled with character defined in fillchar
- fillchar = "string of my choice"
- side = "both", "right", "left"

4 less used equivalent functions to pad()

Series.str.rjust
Fills the left side of strings with an arbitrary character. Equivalent to `Series.str.pad(side='left')`.
Series.str.ljust
Fills the right side of strings with an arbitrary character. Equivalent to `Series.str.pad(side='right')`.
Series.str.center
Fills boths sides of strings with an arbitrary character. Equivalent to `Series.str.pad(side='both')`.
Series.str.zfill
Pad strings in the Series/Index by prepending '0' character. Equivalent to `Series.str.pad(side='left', fillchar='0')`.

Remove ends

strip(), lstrip(), rstrip()

- strip(). - from both ends
- rstrip() - from right
- lstrip() - from left

to_strip = "pattern to remove form end"

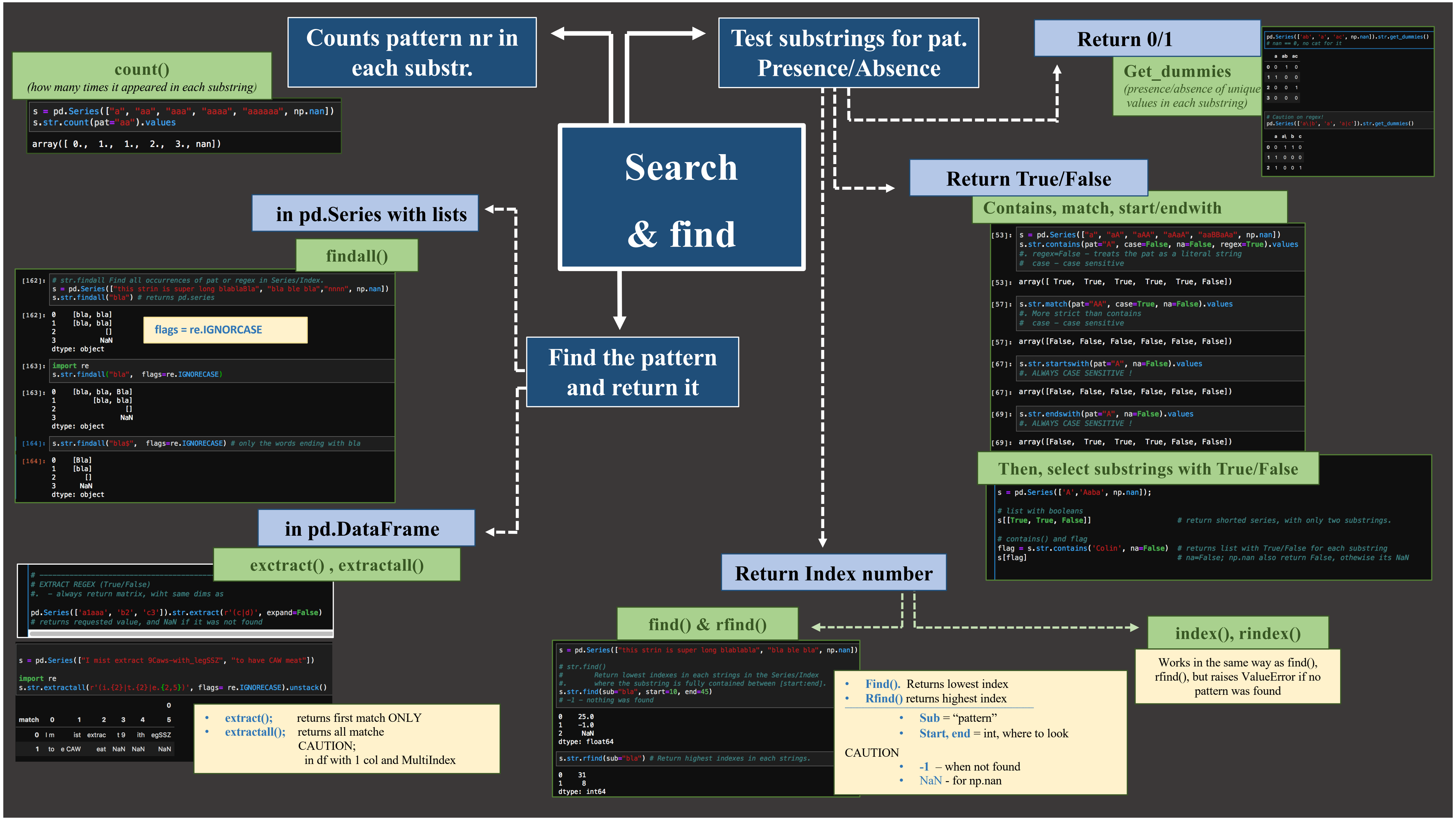
```
[199]: # Remove whitespaces, or new lines
       s = pd.Series(["aBBBa\n", "       \nbCCCb\n", np.nan])
       s.str.strip()

[199]: 0    aBBBa
       1    bCCCb
       2      NaN
       dtype: object

[201]: # Remove specified substring,
       s = pd.Series(["aBBBa\n", "       \nbCCCb\n", np.nan])
       s.str.strip(to_strip="a")

[201]: 0              BBBa\n
       1           \nbCCCb\n
       2                 NaN
       dtype: object
```

https://pandas.pydata.org/pandas-docs/stable/user_guide/text.html



Search & find

Counts pattern nr in each substr.

count()
(how many times it appeared in each substring)

```
s = pd.Series(["a", "aa", "aaa", "aaaa", "aaaaaa", np.nan])
s.str.count(pat="aa").values

array([ 0.,  1.,  1.,  2.,  3., nan])
```

Test substrings for pat. Presence/Absence

Return 0/1

Get_dummies
(presence/absence of unique values in each substring)

```
pd.Series(['ab', 'a', 'ac', np.nan]).str.get_dummies()
# nan == 0, no cat for it
```

	a	ab	ac
0	0	1	0
1	1	0	0
2	0	0	1
3	0	0	0

```
# Caution on regex!
pd.Series(['a\|b', 'a', 'a|c']).str.get_dummies()
```

	a	a\	b	c
0	0	1	1	0
1	1	0	0	0
2	1	0	0	1

Return True/False

Contains, match, start/endwith

```
[53]: s = pd.Series(["a", "aA", "aAA", "aAaA", "aaBBaAa", np.nan])
      s.str.contains(pat="A", case=False, na=False, regex=True).values
      #. regex=False - treats the pat as a literal string
      #  case - case sensitive

[53]: array([ True,  True,  True,  True,  True, False])

[57]: s.str.match(pat="AA", case=True, na=False).values
      #. More strict than contains
      #  case - case sensitive

[57]: array([False, False, False, False, False, False])

[67]: s.str.startswith(pat="A", na=False).values
      #. ALWAYS CASE SENSITIVE !

[67]: array([False, False, False, False, False, False])

[69]: s.str.endswith(pat="A", na=False).values
      #. ALWAYS CASE SENSITIVE !

[69]: array([False,  True,  True,  True, False, False])
```

Then, select substrings with True/False

```
s = pd.Series(['A','Aaba', np.nan]);

# list with booleans
s[[True, True, False]]                      # return shorted series, with only two substrings.

# contains() and flag
flag = s.str.contains('Colin', na=False)  # returns list with True/False for each substring
s[flag]                                     # na=False; np.nan also return False, othewise its NaN
```

Find the pattern and return it

in pd.Series with lists

findall()

```
[162]: # str.findall Find all occurrences of pat or regex in Series/Index.
       s = pd.Series(["this strin is super long blablaBla", "bla ble bla","nnnn", np.nan])
       s.str.findall("bla") # returns pd.series

[162]: 0    [bla, bla]
       1    [bla, bla]
       2            []
       3           NaN
       dtype: object

[163]: import re
       s.str.findall("bla",  flags=re.IGNORECASE)

[163]: 0    [bla, bla, Bla]
       1         [bla, bla]
       2                 []
       3                NaN
       dtype: object

[164]: s.str.findall("bla$",  flags=re.IGNORECASE) # only the words ending with bla

[164]: 0    [Bla]
       1    [bla]
       2       []
       3      NaN
       dtype: object
```

flags = re.IGNORCASE

in pd.DataFrame

exctract() , extractall()

```
# ----------------------------------------
# EXTRACT REGEX (True/False)
#.  - always return matrix, wiht same dims as

pd.Series(['a1aaa', 'b2', 'c3']).str.extract(r'(c|d)', expand=False)
# returns requested value, and NaN if it was not found
```

```
s = pd.Series(["I mist extract 9Caws-with_legSSZ", "to have CAW meat"])

import re
s.str.extractall(r'(i.{2}|t.{2}|e.{2,5})', flags= re.IGNORECASE).unstack()
```

	0					
match	0	1	2	3	4	5
0	I m	ist	extrac	t 9	ith	egSSZ
1	to	e CAW	eat	NaN	NaN	NaN

- **extract();** returns first match ONLY
- **extractall();** returns all matche
  CAUTION;
  in df with 1 col and MultiIndex

Return Index number

find() & rfind()

```
s = pd.Series(["this strin is super long blablabla", "bla ble bla", np.nan])

# str.find()
#        Return lowest indexes in each strings in the Series/Index
#.       where the substring is fully contained between [start:end].
s.str.find(sub="bla", start=10, end=45)
# -1 - nothing was found

0    25.0
1    -1.0
2     NaN
dtype: float64

s.str.rfind(sub="bla") # Return highest indexes in each strings.

0    31
1     8
dtype: int64
```

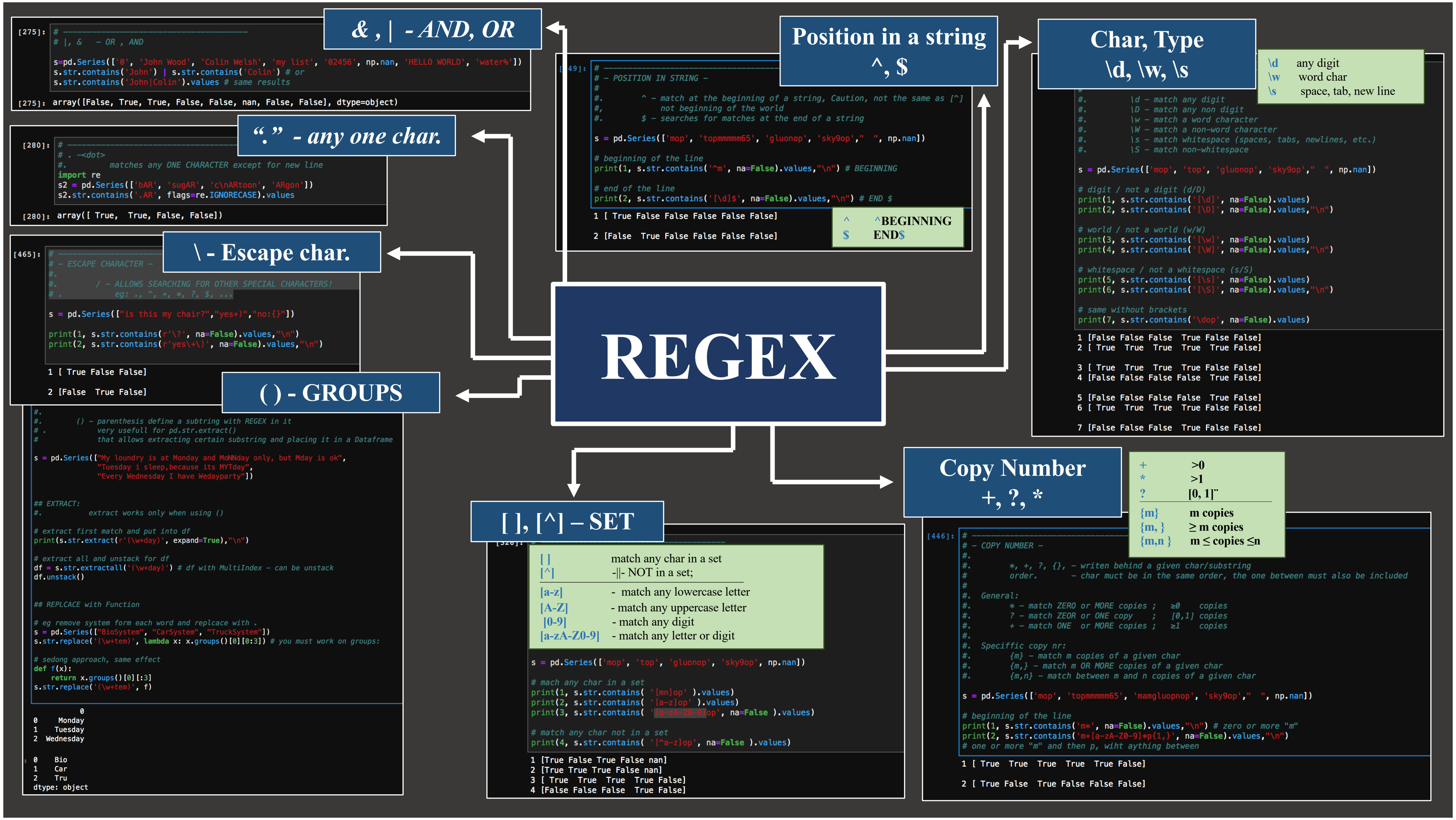
- **Find().** Returns lowest index
- **Rfind()** returns highest index

  - **Sub** = "pattern"
  - **Start, end** = int, where to look

CAUTION
  - **-1** – when not found
  - NaN - for np.nan

index(), rindex()

Works in the same way as find(), rfind(), but raises ValueError if no pattern was found



REGEX

& , | - AND, OR

```
[275]: # ----------------------------------------
       # |, &   - OR , AND

       s=pd.Series(['0', 'John Wood', 'Colin Welsh', 'my list', '02456', np.nan, 'HELLO WORLD', 'water%'])
       s.str.contains('John') | s.str.contains('Colin') # or
       s.str.contains('John|Colin').values # same results

[275]: array([False, True, True, False, False, nan, False, False], dtype=object)
```

"." - any one char.

```
[280]: # ----------------------------------------
       # . -<dot>
       #.          matches any ONE CHARACTER except for new line
       import re
       s2 = pd.Series(['bAR', 'sugAR', 'c\nARtoon', 'ARgon'])
       s2.str.contains('.AR', flags=re.IGNORECASE).values

[280]: array([ True,  True, False, False])
```

\ - Escape char.

```
[465]: # ----------------------------------------
       # - ESCAPE CHARACTER -
       #.
       #.        / - ALLOWS SEARCHING FOR OTHER SPECIAL CHARACTERS!
       # .            eg: ., ^, +, *, ?, $, ...

       s = pd.Series(["is this my chair?","yes+)","no:{}"])

       print(1, s.str.contains(r'\?', na=False).values,"\n")
       print(2, s.str.contains(r'yes\+\)', na=False).values,"\n")

       1 [ True False False]

       2 [False  True False]
```

( ) - GROUPS

```
#.
#.        () - parenthesis define a subtring with REGEX in it
# .            very usefull for pd.str.extract()
#              that allows extracting certain substring and placing it in a Dataframe

s = pd.Series(["My loundry is at Monday and MoNNday only, but Mday is ok",
               "Tuesday i sleep,because its MYTday",
               "Every Wednesday I have Wedayparty"])


## EXTRACT:
#.           extract works only when using ()

# extract first match and put into df
print(s.str.extract(r'(\w+day)', expand=True),"\n")

# extract all and unstack for df
df = s.str.extractall('(\w+day)') # df with MultiIndex - can be unstack
df.unstack()


## REPLCACE with Function

# eg remove system form each word and replcace with .
s = pd.Series(["BioSystem", "CarSystem", "TruckSystem"])
s.str.replace('(\w+tem)', lambda x: x.groups()[0][0:3]) # you must work on groups:

# sedong approach, same effect
def f(x):
    return x.groups()[0][:3]
s.str.replace('(\w+tem)', f)

          0
0     Monday
1    Tuesday
2  Wednesday

0    Bio
1    Car
2    Tru
dtype: object
```

Position in a string ^, $

```
[ 49]: # ----------------------------------------
       # - POSITION IN STRING -
       #
       #.        ^ - match at the beginning of a string, Caution, not the same as [^]
       #,            not beginning of the world
       #.        $ - searches for matches at the end of a string

       s = pd.Series(['mop', 'topmmmmm65', 'gluonop', 'sky9op',"  ", np.nan])

       # beginning of the line
       print(1, s.str.contains('^m', na=False).values,"\n") # BEGINNING

       # end of the line
       print(2, s.str.contains('[\d]$', na=False).values,"\n") # END $

       1 [ True False False False False False]

       2 [False  True False False False False]
```

^	^BEGINNING
$	END$

Char, Type \d, \w, \s

\d	any digit
\w	word char
\s	space, tab, new line

```
#.        \d - match any digit
#.        \D - match any non digit
#.        \w - match a word character
#.        \W - match a non-word character
#.        \s - match whitespace (spaces, tabs, newlines, etc.)
#.        \S - match non-whitespace

s = pd.Series(['mop', 'top', 'gluonop', 'sky9op',"  ", np.nan])

# digit / not a digit (d/D)
print(1, s.str.contains('[\d]', na=False).values)
print(2, s.str.contains('[\D]', na=False).values,"\n")

# world / not a world (w/W)
print(3, s.str.contains('[\w]', na=False).values)
print(4, s.str.contains('[\W]', na=False).values,"\n")

# whitespace / not a whitespace (s/S)
print(5, s.str.contains('[\s]', na=False).values)
print(6, s.str.contains('[\S]', na=False).values,"\n")

# same without brackets
print(7, s.str.contains('\dop', na=False).values)

1 [False False False  True False False]
2 [ True  True  True  True  True False]

3 [ True  True  True  True False False]
4 [False False False False  True False]

5 [False False False False  True False]
6 [ True  True  True  True False False]

7 [False False False  True False False]
```

[ ], [^] – SET

[ ]	match any char in a set
[^]	-||- NOT in a set;
[a-z]	- match any lowercase letter
[A-Z]	- match any uppercase letter
[0-9]	- match any digit
[a-zA-Z0-9]	- match any letter or digit

```
s = pd.Series(['mop', 'top', 'gluonop', 'sky9op', np.nan])

# mach any char in a set
print(1, s.str.contains( '[mn]op' ).values)
print(2, s.str.contains( '[a-z]op' ).values)
print(3, s.str.contains( '[a-zA-Z0-9]op', na=False ).values)

# match any char not in a set
print(4, s.str.contains( '[^a-z]op', na=False ).values)

1 [True False True False nan]
2 [True True True False nan]
3 [ True  True  True  True False]
4 [False False False  True False]
```

Copy Number +, ?, *

+	>0
*	>1
?	[0, 1]¨
{m}	m copies
{m, }	≥ m copies
{m,n }	m ≤ copies ≤n

```
[446]: # ----------------------------------------
       # - COPY NUMBER -
       #.
       #.        *, +, ?, {}, - writen behind a given char/substring
       #         order.       - char muct be in the same order, the one between must also be included
       #
       #.  General:
       #.        * - match ZERO or MORE copies ;   ≥0    copies
       #.        ? - match ZEOR or ONE copy    ;   [0,1] copies
       #.        + - match ONE  or MORE copies ;   ≥1    copies
       #.
       #.  Speciffic copy nr:
       #.        {m} - match m copies of a given char
       #.        {m,} - match m OR MORE copies of a given char
       #.        {m,n} - match between m and n copies of a given char

       s = pd.Series(['mop', 'topmmmmm65', 'mamgluopnop', 'sky9op',"  ", np.nan])

       # beginning of the line
       print(1, s.str.contains('m*', na=False).values,"\n") # zero or more "m"
       print(2, s.str.contains('m+[a-zA-Z0-9]*p{1,}', na=False).values,"\n")
       # one or more "m" and then p, wiht aything between

       1 [ True  True  True  True  True False]

       2 [ True False  True False False False]
```